NETWORKED VAX/LSI/CAMAC DATA ACQUISITION SYSTEM DEVELOPMENT

J.D. Melvin, M.H. Mendenhall, T.A. Tombrello
California Institute of Technology 301-38
Pasadena, California 91125

D.L. Clark
Nuclear Structure Research Laboratory
University of Rochester
Rochester, New York 14627

Summary. Recent development of the Caltech data acquisition system installed in 1981, which runs on a VAX-11/750, a Peritek Q-bus network, LSI-11s, and CAMAC, is described. In this system, the DEC VMS and RT-11 operating systems are supported on the VAX "host" and LSI-11 "front-end" computers by a VMS device driver and network host program, and a bootable RT-11 device driver. Network "utility" and "control" programs provide general purpose support for communication between front-end and host software. Data acquisition software tools are provided for writing programs to run nuclear physics experiments. A system similar to Caltech's was installed at the University of Rochester in 1982. The network has been tested for speed and real-time response. After including all software overhead required by data acquisition, it was found that the system could transfer buffers and acknowledge their receipt at a net speed of 127 KB per second with a 35% load on the host computer. The network software is currently being rehosted on Ethernet hardware at Caltech in a multiple host - many front-end computer configuration. Compatibility with the current Peritek network software will be maintained.

Introduction

Upgrade of the nuclear data acquisition capability at Caltech was started in 1979. From 1979 to 1981 commercial and research-laboratory developed systems in various parts of the USA were examined, and in 1981 it was decided that the most promising approach for Caltech would be to develop system of our own design. This project was initiated under the direction of the first author in 1981. It has grown into an inter-divisional (Physics/Engineering) and inter-university (Caltech/University of Rochester) effort because of the general instrumentation problems it has addressed.

The overall objective of the Caltech/Rochester project is to develop tools with which any given data acquisition and analysis task can be implemented easily on several networked computers, each optimized for a different part of the over-all task. Unlike many current networked systems, the focus of the computer network is on speed and real-time response.

In the following sections we present first the history of the Caltech/Rochester project. We then describe in functional detail the network hardware and software, utility software, and data acquisition software tools developed. Next we present results of a speed test on the network, and after that we comment on the current effort to rehost the network on the new Ethernet hardware. Finally, in the Appendix, we give technical details of network software developed to date.

Project History

Data Acquisition Hardware

Around 10 years ago Caltech augmented its hardwired data acquisition systems with a programmable system based on a computer. Because older general purpose computers carried out the basic data acquisition tasks of event handling and histogramming much more slowly than hardwired systems, this early system was built around a special purpose computer. Over the years we gradually discovered that this computer was practically impossible to reprogram. It became clear that special purpose computers, because of their small market, would never support operating systems and compilers comparable to those becoming available on general purpose computers.

During the past six years general purpose computers have become faster and less expensive, and operating system software has developed considerably. The recent supermini computers, such as the VAX (which we use now), offer excellent program development software, virtual memory (which allows computations on very large data arrays), and concurrent support of multiple users. Although these new computers have computational speed adequate for the analysis part of data acquisition, they still handle events too slowly. We therefore decided (as have many laboratories) to divide data acquisition tasks between (1) specialized "front-end" computers which handle events quickly, and (2) easy to program "host" computers that can analyze buffers full of events quickly. By this division, we hoped to take maximum advantage of commercially available hardware and software.

At Caltech and at the University of Rochester DEC computer hardware and operating systems were selected because of their reliability and because of their wide use at our universities and by the nuclear physics community as a whole. VAX-11/750 computers were selected as host computers, and LSI-11/2 and LSI-11/23 computers as front-end computers. (Faster front-end computers are available but are much more difficult to program.)

Various links between the front-end and host computers are possible. We found that dedicated links connecting two computers were most common. Such links use various standard or custom parallel or serial interfaces. In some cases, the front-end computer used was a programmable CAMAC branch driver, and the computer link was the CAMAC interface.

We decided to consider the alternative more general approach of the computer network. Computer networks connect several computers, often over a common cable, and support a variety of inter-computer communication.

0018-9499/83/1000-3816$01.00

Currently available networks (for example DECNET) are slow and/or expensive. Therefore, we selected the low cost, fast Peritek HEX-11 network hardware and wrote our own software.

A hardware system was assembled in the W.K. Kellogg Radiation Laboratory at Caltech in early 1981 by the first two authors. This initial hardware system is accurately described in our previous paper[1] (except that the VAX disk is now a Systems Industries disk on the internal VAX bus -- off the Unibus). A hardware system similar to Kellogg's was assembled at the University of Rochester by the first and fourth authors during the spring and summer of 1982. It is described in a separate paper also being presented at this conference [2].

During 1983 the network will be rehosted by the first author on the recently available, faster, and more flexible Ethernet network hardware. This rehosting is a joint effort of members of the Physics and Engineering Divisions at Caltech.

Network Software

General purpose network software has improved greatly during the last few years as programmers have developed methods for organizing the complex communication problems that occur between independent computers. Such software is oriented towards:

1. Sending messages between networked computers;
2. Allowing computers on a network to use each other's resources;
3. Handling accounting and authorization for resource use and file protection across several systems;
4. Rerouting network messages when a link of the network is not serviceable.

Solutions to these problems are often complex, and have made network software costly and frequently much slower than the hardware.

At Caltech we decided to write our own network software. No commercial software would allow us to achieve our goals of speed and real-time response for the network at reasonable cost. In designing the software, we tried to maximize use of software interfaces and data structures in standard computer operating systems. As a result a network software system was completed rather quickly by the first two authors and was found to have more than adequate speed and flexibility for data acquisition. The current network software, which is described in this paper, differs in detail from that outlined in our previous paper[1] but upholds the same design goals.

Data Acquisition Software

Once the network software was written, some general network related utilities were written, and then attention was turned to data acquisition software. It was decided to approach this software also in a way which would take maximal advantage of software tools in standard computer operating systems. Our approach is explained in the following paragraphs.

In the days of hardwired analyzers, setup and control was carried out by pressing buttons on instrument panels. In many data acquisition software systems which we studied, this approach was imitated: One was furnished with a large number of "buttons" in the form of typed commands which allow for experiment configuration and operation, and often these commands could be chained together for automatic experiment control. (Of course, the new data acquisition systems, being programs, had the great advantage that they could be modified for those rare experiments which were not handled by the standard systems.) Such command driven systems offered great ease of use, but also required a large programming effort, beyond that which we could support.

The development of more flexible, easier to program computers has made another approach to data acquisition programming practical. The experimenter could be provided with a good commercial operating system for program development and data acquisition specific software tools for implementing experiments. The software tools would be designed to allow the full range of experiments to be handled, from the simplest ones of single parameter acquisition and analysis, to complex ones including multiple parameters and fed-back experiment control, beyond the capability of a command driven system. The experimenter could also be provided with program templates showing how to apply the software tools to common data acquisition problems. The essence of this approach to data acquisition software is that use of programming tools and command structures in commercial operating systems is maximized, and the effort of including such structures within a data acquisition software system, which can only be done with sacrifice of flexibility and/or a large programming effort, is minimized.

Data acquisition software based on this approach and which utilized the network we developed was drafted by the first author starting in late 1981. This software has been developed further by other Kellogg personnel at Caltech and various University of Rochester personnel during 1982 and 1983.

Graphics

The network system developed at Caltech and the University of Rochester would clearly support complex interactive graphics as well as high speed data acquisition. Both kinds of processing involve fast front-end response to events, buffer transfers, and processing power and flexibility that is best implemented on a host (larger) computer. In graphics processing, the events are requests for display change and graphics device control interrupts. Buffers are data to be graphically displayed which are sent from the host to the front-end computer. The host provides the computational power to extract display data from experiment data.

At the University of Rochester the front-end computers contain small array processors (made by Sky Computer)[3]. This offers a useful option for programmable control of graphics displays: Such array processors could be used to translate, rotate, or otherwise transform displays faster than a VAX host could, thus off-loading from the host simple but computationally intensive tasks. It is planned that these array processors also carry out simple, computationally intensive data analysis in nuclear physics experiments.

At Caltech the capability of the network to support complex graphics came to the attention of Engineering Division personnel. As a result, two VAX computers and some salary support were made available for rehosting of the network software system on the new Ethernet hardware. Ethernet hardware allows computers separated by as much as 500 m to communicate. The new Caltech network will span three buildings and will support graphics work stations for education of students in computer-aided engineering concurrently with use for data acquisition.

Network Hardware

Many kinds of local area network hardware are now available. Some communicate data in parallel (several bits per clock pulse), as does the Peritek network (which we currently use), but are usually limited to inter-computer distances of a few tens of feet. Others convert the data into a serial stream which can be transmitted much farther. Ethernet (which we will use soon) is a network of the latter design, and will probably become the standard of the computer industry; it is presently accepted by Xerox, DEC, and Intel as their future standard.

The Caltech network installed in 1981 uses Peritek hardware. During 1983 the Caltech network will be rehosted on Ethernet hardware built by Interlan.

A description of the Peritek and Ethernet hardware follows:

Peritek Hardware

Peritek hardware costs about 20% as much as Ethernet hardware per computer (around $500 per computer), and offers speeds up to 163 KB per second (see the section on network speed below). It is restricted to a 40 foot inter-computer separation. One computer serves as the network master (the host computer in our systems) -- all communication is arbitrated by and must pass through the network master. Each front-end computer is connected by a separate 40 line ribbon cable to the network master, and connections are direct with no electrical isolation. Front-end computers can be connected and disconnected from their cables while the network is running.

Ethernet Hardware

Ethernet hardware allows up to a hundred computers to communicate on each Ethernet cable segment, which may be up to 500 m long. Several cable segments may be interconnected by point-to-point links up to 1000 m long. No computer serves as a network master (except as desired in software); each Ethernet interface handles arbitration for the cable, message transmission, and retries if messages collide autonomously. Any interface may direct communication to any other interface. All communication occurs on coaxial cable with several hundred volts of transformer isolation. Raw speed on Ethernet is 1.25 MB (10 megabits) per second. Actual speed is about a third of this because messages are buffered in the Ethernet interfaces. (We note that one could modify the Peritek hardware with FIFO buffers and transmitter-receivers to outperform Ethernet in speed and distance, although not in generality or flexibility.) Ethernet interfaces may be installed on and removed from the Ethernet cable while the network is running.

The Peritek network hardware operates on the Q-bus of the LSI-11 front-end computers that we are using. The particular Ethernet hardware that we have selected is built by Interlan, also for the Q-bus. On all VAX computers we have installed Q-bus extensions to the Unibus by using Able Quniverters; the hosts' network interfaces reside on these Q-bus extensions.

Both Peritek and Ethernet (Interlan) network hardware are adequate in speed for nuclear data acquisition, which generally requires transfer rates below 30 KW (60 KB) per second. For a small laboratory not requiring long runs between host and front-end (no front-end computer in the target area!), Peritek is the most cost effective choice.

The Ethernet project at Caltech has different requirements and goals from those of a typical small nuclear physics laboratory: Data acquisition front-end computers located near a tandem accelerator in one building will be connected on a 500 m Ethernet cable segment to two VAX computers in two other buildings. These computers, in addition to data acquisition, run computer - aided engineering software and are expected to use the network concurrently for non data acquisition activities. Personnel of both the Physics and Engineering Divisions are involved in this Ethernet project, as both divisions have computational needs in data acquisition and graphics. The project represents a feasibility study to demonstrate that high speed, real-time computational needs can be satisfied by a network of distributed resources. (It should be noted here that there is currently a parallel campus-wide Caltech effort to provide resource sharing and terminal hookups using Ethernet technology. We anticipate that the Physics / Engineering network described in this paper will be integrated into the campus network by direct connection, connection through a gateway, or connection through one or more computers which have nodes on each network.)

Software

The Caltech software is written in layers in order to facilitate transportability and modularity. The first layer is specific to network hardware. A second layer provides general utilities for carrying out software tasks shared between host and front-end computers. A third layer supports nuclear data acquisition. The second and third layers are network hardware independent, and communicate with the first layer through standard software constructs of the VMS and RT-11 operating systems. We describe these software layers starting with the first layer.

Network Software Layer

The Caltech network software implements intercomputer communication required by high speed data acquisition. Specifically, we have chosen to support:

1. Sending messages from the host to the front-end computers;
2. Sending buffers between the host and front-end computers (data buffers go to the host, graphical and extensive control information go to the front-end).

In implementing item 2, we supported one other very useful kind of network communication: Emulation of disk transfers for the front-end computer on the network. This capability was developed with the help and experience of the second author with computer networks at Washington University in St. Louis. It allows low cost, disk-less, front-end computers to run complete disk-based operating systems. Also, it allows us to take maximum advantage of commercial software structures: Instead of introducing a special message protocol for network communication, we draw on the hardware independent disk read-write call structure offered by the front-end computer operating system (RT-11 in our case).

"Pseudo-disks" for the network disk emulation are set up within virtual memory arrays of a network "host" program on the VAX. Because of the way the VAX VMS operating system handles virtual memory, the pseudo-disk transfers are cached in VAX memory. The resulting disk emulation performance often exceeds that of a hard disk because the latter requires seek time to locate data.

Utility Software Layer

In addition to the basic network software that allows communication of messages and data, a layer of network utilities has been written which supports the following data acquisition relevant software actions:

1. Control of programs in the front-end computer by programs on the host computer and vice-versa;
2. Awakening of a program on the host computer each time a special data buffer transfer occurs from the front-end computer into the host. (When the data acquisition host program is not executing, the multi-user host can service other programs. In fact, several data acquisition programs with separate front-end computers can run concurrently on a single host without writing special software).

The above actions are carried out rapidly, with guaranteed timing, so that an experiment can be controlled properly and so that buffers of acquired data are analyzed before the buffers are needed for new data.

In addition to these actions, the following convenience features are supported by the network utility software:

3. Initiation of data acquisition from any host terminal in such away that the acquisition can continue autonomously, independent of subsequent terminal usage;
4. Connecting to and disconnecting from on-going data acquisition from any terminal;
5. Common data areas in the host (generally used for histogramming) which can be concurrently accessed by the data acquisition program and by experiment display and control programs. Common data areas allow one or more people to examine one on-going experiment, and allow an experimenter to write, compile, and run display and analysis programs that act on incoming data while an experiment is running.

Data Acquisition Software Layer

The above utility software is described in terms of data acquisition, but could just as well be used to support any software task shared between host and front-end computers. Outside of the utility software layer lies a third software layer specific to data acquisition. Part of this third software layer must be tailored to the specific data collection hardware of the laboratory that uses it. For example, Caltech uses Kinetic Systems memory mapped CAMAC crate controllers while the University of Rochester uses Jorway parallel/serial CAMAC highway branch drivers with DMA interfaces.

As discussed above, we have not written a complete, command driven data acquisition system. Instead, the experimenter uses the software development tools of the operating system and is provided with data acquisition specific software tools in the form of (1) macro and subroutine libraries, and (2) selected programming languages.

Macros and subroutines are provided to implement the following basic data acquisition related software actions:

1. Transfers to and from data collection interfaces, e.g. CAMAC (macros, subroutines);
2. Filling of buffers with collected data (macros, subroutines);
3. Sending of filled buffers to a data analysis program (subroutines);
4. Communication of messages between the collection and analysis programs (subroutines).

Macro assembly language and Fortran languages are provided for coding:

1. Response to real-time events (nuclear events in detectors), which response normally consists of reading data acquisition interfaces, e.g. CAMAC, and storing data in a buffer (Macro);
2. Analysis of buffers, including gating and histogramming (Fortran);
3. If needed, experiment control based on feedback from data collected, elapsed time, etc. (Fortran).

The software is structured so that distribution of the above tasks between front-end and host computers hardly need concern the experimenter. Both computers support similar Macro assemblers and Fortran compilers, and message and buffer transfer subroutines handle all network communication transparently.

Network Speed

There are two speeds in the described network software system which are important for nuclear data acquisition. One is the speed at which data buffers are transferred from a front-end computer to the host. The other is the speed at which a signal from a front-end computer (for readers of the Appendix, write

to block 0 of NT7) can pass through all software layers (NTDRIVER, NETHOST, and NETUTIL) to awaken a host program, and can cause that program to signal back to the front-end computer (write to SATmNETMB which causes NETHOST to send an unsolicited message over the network). We call this the "turn-around message" speed, and designate turn-around message by TAM for short.

A simple benchmark was written: The LSI-11 program for the benchmark carries out a pseudo-disk write (buffer transfer), followed by a VAX program wake-up signal (write to block 0 of NT7), and then program loops waiting for a return message from the VAX (waiting for the command word to be filled by an unsolicited network message). When the return message arrives, the program starts over, repeating the above operations. The corresponding VAX program for the benchmark calls the library subroutine which awaits a wake-up message from the LSI-11 (mail box message from NETUTIL initiated by NETHOST when a write by the LSI-11 to block 0 of NT7 is received) and then sends a message back (writes a command to SATmNETMB which is transmitted by NETHOST over the network as an unsolicited message). 2000 iterations of these programs were timed for different sizes of buffers transferred. The results are as follows:

LSI-11/2 FRONT-END

Buffer size (bytes)	Time for 2000 cycles	Conclusion
0	73 sec	37ms / TAM
1024	104 sec	37ms / TAM + 65 KB/sec buffer transfer rate (Effective transfer rate: 19 KB/sec)
16384	284 sec	37ms / TAM + 150 KB/sec buffer transfer rate (Effective transfer rate: 113 KB/sec)

LSI-11/23 FRONT-END

Buffer size (bytes)	Time for 2000 cycles	Conclusion
0	58 sec	29ms / TAM
1024	84 sec	29ms / TAM + 77 KB/sec buffer transfer rate (Effective transfer rate: 24 KB/sec)
16384	252 sec	29ms / TAM + 163 KB/sec buffer transfer rate (Effective transfer rate: 127 KB/sec)

During the operation of this benchmark, the time in user mode on the Kellogg VAX-11/750 dropped from around 80% to around 45%. Thus, we estimate that about 35% of the VAX CPU time was being used for network operation. Based on this measurement we compute that for 30 KW/sec (60 KB/sec), the maximum reasonable rate in a nuclear physics experiment, and for 8 KW buffers, 60/127 x 35% = 17% of the VAX CPU time would be used to support the network transfers.

Parenthetically we add that the VAX-11/750 can histogram, flat-out, 75000 single parameter events per second. Thus an experiment where 30000 events per second are acquired by a front-end computer, transmitted over the network in 8 KW buffers, and histogrammed would use 17% (network) + 30000/75000 (histogramming) = 57% of the VAX CPU time. Of course, this is poor use of the power of a VAX!

At the University of Rochester a more complete benchmark was written that included the utility software layer (SETUP and CONTROL), and passed buffers and messages between the LSI-11 and the VAX with the data acquisition library routines provided (see above). The result for two thousand 1024 byte plus TAM transfers using an LSI-11/23 was 84 seconds, identical to the Caltech result. Thus, the utility and data-acquisition software layers add insignificant execution time to the system.

Rehosting on Ethernet

The rehosting of the network on Ethernet requires rewriting the RT-11 NT device driver, and the VMS NTDRIVER and NETHOST programs (see the Appendix for a description of NT, NTDRIVER, and NETHOST). By preserving all mail box and event flag interfaces, all other software layers developed are expected to run without change.

Because of the greater flexibility of Ethernet as compared to the Peritek network, some enhancements of the system are planned. One enhancement is the ability to select one of several hosts with which network transfers are to occur from a given front-end computer. One application of this capability is that two copies of the RT-11 network device driver could be loaded into the front-end computer and directed to two different hosts for simultaneous operation with the two hosts. Another application is that if one host computer goes down, an on-going experiment could be continued with another host. A second planned Ethernet enhancement is the elimination of the serial lines between front-end and host computers by adding a "pseudo-terminal" communication channel to the network software. This pseudo-terminal communication would be added to the RT-11 network device driver and VAX network host programs in a manner which operated asynchronously with respect to pseudo-disk and message transfers. Using this channel it is expected that each computer will be able to emulate terminal communication on the other.

Appendix: Technical Description of the Software

Network Software Layer

Two kinds of network communication are supported. One kind is disk emulating reads and writes controlled by the front end LSI-11 computer. This communication consists of DMA transfers of 1 to 8 K word buffers. The other

kind of communication is transmission of a two word message from the VAX host to an LSI-11 by interrupt and programmed handshake under VAX control. This kind of communication is called "unsolicited network message", unsolicited because is is not initiated by the LSI-11.

We have written a bootable RT-11 device driver NT to control the Peritek network hardware from the LSI-11 end. On boot, ROM code on the network interface signals the host to DMA a boot block into LSI-11 memory. This boot block has code to boot the RT-11 operating system. The RT-11 device driver supports pseudo-disk reads and writes to eight network pseudo-disk units, NT0, NT1, ..., NT7. Besides reads and writes, the device driver supports two special function calls. One call is the standard RT-11 call for variable size devices; it returns the device size of the unit specified. The other call supplies addresses of two words, called "command" and "flags" words, and (optionally) of an interrupt routine, which are used as follows when an unsolicited message comes over the network from a program running on the VAX: The network message contains two words, one of which is loaded into the command word in LSI-11 memory provided that this word is currently zero (which signifies that the LSI-11 program has picked up the previous command word). The other message word is or-ed into the LSI-11 flags word. Typically, the command word is used for control of an LSI-11 program by a VAX program, and the flags word is used to signal to the LSI-11 program when the VAX has finished with a buffer. After the message words are loaded, the supplied interrupt routine (if any) is called.

On the VAX, a device driver NTDRIVER was written which fields interrupts from LSI-11 computers requesting network transfers, and handles network DMA read and write transfers. All other network control is carried out in a Fortran host program NETHOST which runs at priority 15. NETHOST manipulates the network hardware to (1) obtain specific network transfer requests from the LSI-11s and (2) return transfer status to the LSI-11s.

In addition to network hardware control, NETHOST maintains global sections which act as the pseudo-disks for the LSI-11s. NETHOST communicates through a VAX mail box NETHOSTMBX with a program HS ("host-set") which is run by a system operator to map and unmap pseudo-disks from global sections. Flexible pseudo-disk mapping is supported: A global section may be mapped to one or more pseudo-disk units of one or more LSI-11s so that each LSI-11 has unit specific read or read-write access to each global section. In addition, programs have been written for the VAX which allow one to directory and copy RT-11 files from within pseudo-disk format global sections.

Note: A modification was made to RT-11 on the LSI-11s so that all console commands draw utility programs, compilers, and libraries from a device with logical name SL rather than from SY. One pseudo-disk which contains all of the above mentioned software is mapped with read-only access to NT1 of all LSI-11s, and NT1 is assigned to SL. Thus, duplicate pseudo-disk space with RT-11 utility programs, compilers, and libraries need not be provided for multiple LSI-11s.

NETHOST also provides the following services: On a pseudo-disk write to block zero of NT7, a flag in VMS event flag cluster NET7FLAGS is set to awaken the utility program NETUTIL and inform it which LSI-11 did the write (see below for action of NETUTIL). NETHOST watches for input from mail boxes SATmNETMB where m is the number of the front-end LSI-11 computer, often called a "satellite". When a message arrives, two words are extracted from it and sent to the indicated LSI-11 as an unsolicited message over the network.

Utility Software Layer

A Fortran program NETUTIL was written which runs on the VAX at priority 6. It awaits a signal from NETHOST in NET7FLAGS that tells it when an LSI-11 writes to block zero of pseudo-disk unit NT7. NETUTIL has arrays mapped to the global sections used by NETHOST for NT7 of each LSI-11. After receiving a NET7FLAGS signal, NETUTIL picks up the first null terminated character string in block 0 of NT7 and creates a VAX process running the program in the file given by that string if such a process has not already been created. It then sends a 16 character mail box message to that process, the first 12 character giving the user name of the owner of the LSI-11 (see below) and last four characters being "SAT"m where m is the LSI-11's number. This message is typically used to awaken a VAX data analysis program waiting for new data buffers from an LSI-11 data collection program. (Usually, return messages from the VAX data analysis program to the LSI-11 data collection program are sent by writing to mail box SATmNETMB. As described in the previous section, NETHOST picks up such messages and sends them as unsolicited messages over the network.)

LSI-11 ownership is established as follows: Each LSI-11 has a serial line connecting it with the VAX. This serial line must be allocated by (and therefore owned by) a VAX user prior to any LSI-11 operation which invokes NETUTIL or else an error message will be delivered to the LSI-11. The owner of the serial line is considered the owner of the LSI-11 by NETUTIL.

Another utility program, CONTROL, was written to control LSI-11 operation from a detached process on the VAX and support communication between this detached process and an interactive user (user at a terminal). CONTROL is operated as follows: The experimenter logs on to the VAX and runs a privileged setup program which starts a detached process under his or her account running the program CONTROL. CONTROL is told the number of an LSI-11 and the UIC (user identification code) of the experimenter who started CONTROL. CONTROL allocates the LSI-11 serial line and thereby becomes owner of the LSI-11. Interactive user initiated messages to be sent to the LSI-11 over the serial line or via unsolicited network messages are subsequently routed through CONTROL: An interactive user may communicate with CONTROL by sending messages through VAX mail box SATmCTRLMB (m=LSI-11 number). Communication acknowledgement is sent through event flags SATm. If the user has the same UIC as CONTROL, the user may connect to CONTROL with a connect

message. CONTROL saves the PID (process identification) of the user. Thereafter, until the user logs off of the VAX, the user may issue requests to CONTROL to send messages to the LSI-11. The PID sent with any requests is checked by control, and if it is the connected user's PID, the messages are forwarded to the LSI-11.

CONTROL serves several other useful functions. The VAX program created by NETUTIL to communicate with an LSI-11 program is not attached to a terminal. CONTROL allows this program to communicate with a user who is connected to CONTROL as follows: Mail boxes SATmMESSMB are provided on the VAX for each LSI-11. A write to the mail box corresponding to the LSI-11 that CONTROL is operating causes CONTROL to check if there is a user connected to itself. If so, the message is broadcast to that user's terminal, appearing on the terminal screen regardless of the program the user is currently running.

If no user is connected to CONTROL, the message is saved in a file. When a user subsequently connects to CONTROL, he or she immediately receives all pending messages. CONTROL logs all such messages to the user as well as all user requested messages to be sent to an LSI-11 in a file.

Data Acquisition Layer

A program SETUP is provided which accepts from an input file:

1. The name of an LSI-11 front-end program;
2. The data buffer size and degree of multibuffering in the LSI-11 and VAX memories;
3. Code for the VAX data analysis program which is to communicate with the front-end program.

The experimenter must previously have created, compiled, and linked with data acquisition subroutine and macro libraries the LSI-11 program. The SETUP program (1) applies a preprocessor (discussed below) to the VAX code, (2) compiles and links this code with a VAX subroutine library, (3) starts a process under the account of the experimenter which runs the program CONTROL, (4) directs CONTROL to to allocate the serial line to the LSI-11 thereby taking ownership of the LSI-11, and via CONTROL, (5) starts the LSI-11 program by sending "RUN <LSI-11 program name>" over the serial line, and (6) sends to the LSI-11 program the name of the VAX data analysis program with which it will communicate as well as (7) the specified buffer size and degree of multibuffering. The LSI-11 program begins by calling a setup subroutine on the LSI-11. This setup subroutine sets up data areas for buffers, and writes to block 0 of NT7 the name of the VAX data analysis program with which it is to communicate. On receiving this write to block 0 of NT7, NETHOST flags NETUTIL and NETUTIL starts the specified VAX program. The LSI-11 program then sends a message which is received by the VAX program it started, and this VAX program passes the message on through SATmMESSMB to CONTROL and thereby to the experimenter's terminal, informing the experimenter that all programs are running and that data acquisition can start. The VAX and LSI-11 programs will thereafter communicate with each other by:

1. Pseudo-disk reads and writes for data transfer;
2. Writes by the LSI-11 program to block 0 of NT7, which cause NETHOST to signal NETUTIL to send a mail box message to the VAX program;
3. Writes by the VAX program to SATmNETMB, which get passed by NETHOST to the LSI-11 program via unsolicited network message.

The LSI-11 and VAX programs are quite flexible and can easily be tailored to specific experiment needs. The only restrictions are that the programs must start by calling library setup subroutines, and that they must pass buffers and messages to each other and to user terminals with the library subroutines provided.

In order that the VAX data acquisition program store analyzed data in a form that may be simultaneously accessed by other programs, a global section SATmUTIL is provided for each LSI-11. The VAX code supplied as input to SETUP may include "special declarations" SPECTRUM and WINDOW for arrays. These declarations are converted by a Fortran preprocessor in SETUP into integer*4 and byte array declarations, and the arrays so declared are equivalenced to parts of a memory region mapped to SATmUTIL and are pointed to by directory entries of Caltech design within SATmUTIL. Other VAX programs can map to SATmUTIL and, by using the directory structure, can locate these same arrays by name.

References

[1] VAX/LSI-11/CAMAC Nuclear Data Acquisition System under Development at the W.K. Kellogg Radiation Laboratory, Caltech, J.D. Melvin, M.H. Mendenhall, R. McKeown, and T.A. Tombrello, IEEE Trans. on Nucl. Sci. NS-28, 3739 (1981).

[2] VAX-11 Based Acquisition Computer System at the Nuclear Structure Research Laboratory, D.L. Clark, T.S. Lund, and J.D. Melvin. Paper presented at the Biennial Conference on Real-Time Computer Applications in Nuclear and Particle Physics, Berkeley, CA (May 16-19, 1983).

[3] The Performance of an LSI-11/23 with Array Processor as High Speed Front End Processor, D.L. Clark. Paper presented at the same conference as reference [2].

Acknowledgements

Thanks are due to the following contributors to the systems described in this paper: At the W.K. Kellogg Radiation Laboratory, Caltech, K. McLoughlin, D. Potterveld, R. McKeown; in the Engineering Division at Caltech, P. Dimotakis; at the Nuclear Structure Research Laboratory, University of Rochester, T. Lund and P. Blazer.

This project has been supported in part by NSF grants PHY79-23638 at the California Institute of Technology and PHY79-23307 at the University of Rochester. The rehosting of the network on Ethernet has also been supported in part by the Engineering Division of the California Institute of Technology.